# Bases de données relationnelles

## 1. Idée générale

Une **base de données relationnelle** organise les données dans des **tables** (appelées *relations*) reliées entre elles. L'objectif est d'éviter la redondance, de garantir la cohérence, et de permettre des requêtes précises sur des données structurées.

- **Pas de redondance :** une même information ne doit pas apparaître dans plusieurs tables ; sinon, on risque des oublis ou des ratés lors de mises à jour, insertions ou suppressions. Une information répétée dans une même table est aussi souvent le signe d'un mauvais schéma.
- **Cohérence :** toutes les données doivent avoir un type « adapté », et être stockées « dans le même format ».
  **Exemple d'incohérence :** stocker une date tantôt sous la forme *numérique* `20260520`, tantôt sous la forme d'une *chaîne de caractères* `"20260520"`. Cela peut occasionner des bugs ultérieurs, lors d'un traitement appliqué à ces données...

## 2. Modèle relationnel : vocabulaire

| Terme | Signification |
|---|---|
| **Relation** | Une **table** : *ensemble de lignes*, structurées selon les mêmes *colonnes*. |
| **Attribut** | Une **colonne** de la *table* (ex. : `nom`, `age`). Chaque colonne doit avoir un *domaine*. |
| **Domaine** | Le **type** de valeurs admissibles pour un attribut (entier, texte, date...). |
| **p-uplet** (ou *tuple*) | Une **ligne** de la table : on parle aussi d'**enregistrement**. |
| **Schéma relationnel** | La description (codifiée) d'une relation : son nom, ses attributs et leurs domaines. |

💭 **Remarque : tuple ou liste ?** ▼

On emploie souvent les *tuples* dans ce contexte, plutôt que des listes : cela peut surprendre. Une ligne d'une table est un « paquet de valeurs » provenant d'un enregistrement. Modifier ces valeurs stockées dans une *liste* n'aurait **aucun impact** sur les valeurs stockées dans l'*enregistrement* dont elles sont issues (il existe un autre mécanisme pour cela). Utiliser des tuples aide à *ancrer* cette idée…

# 3. Contraintes d'intégrité

## 3.1. Contrainte de domaine

Chaque valeur d'un attribut doit appartenir au domaine défini pour cet attribut. Un attribut `age : INT` ne peut pas contenir la chaîne `"inconnu"`.

💭 **Remarque : 50 nuances de types** ▼

Il faut souvent être plus précis que « `int` » ou « `str` ». Beaucoup de systèmes de gestion de bases de données exigent par exemple qu'on dise *combien de caractères* ***au maximum*** comportera une donnée de type « chaîne de caractères ».

## 3.2. Contrainte de relation — clef primaire

Une **clef primaire** est un attribut (ou un ensemble d'attributs) dont la valeur identifie de façon **unique** chaque p-uplet. Elle ne peut pas être `NULL`.

💭 **Remarque : NULL ?** ▼

`NULL` représente une *absence* d'information. C'est cette « valeur » qu'on verra lorsqu'une information **n**'a **pas** été renseignée dans la base : c'est parfois possible… mais parfois non. Selon la façon dont la base de donnée est définie, certaines informations **devront *impérativement*** être fournies : si ça n'est pas le cas, le système de gestion de bases de données refusera d'accepter un nouvel enregistrement incomplet.

On signale la clef primaire en la **soulignant** dans le *schéma relationnel*.

✏️ **Exemple : un exemple concret**

Imaginons qu'on veuille créer un carnet de notes. Il faut commencer par prévoir une relation (une table) pour enregistrer chaque élève. Pour pouvoir gérer des élèves portant les mêmes nom et prénom au sein d'une même classe, on prévoira une clef primaire numérique, qui devra absolument être renseignée, et qui augmentera de 1 à chaque nouvel élève ajouté dans la table. On décrira alors cette table en écrivant la relation :

**`Eleve(id_eleve : INT, nom : TEXT, prenom : TEXT, classe : TEXT)`**

## 3.3. Contrainte de référence : clef étrangère

Une **clef *étrangère*** est un attribut (un « nom de colonne ») qui contient, pour chaque ligne, l'identifiant d'une ligne d'une *autre table*.

On la représente **précédée d'un** `#` dans le *schéma relationnel*.

✏️ **Exemple : un exemple concret (suite)**

Poursuivons avec le carnet de notes. On doit prévoir une table pour enregistrer les notes, *toutes* les notes. On pourra alors décrire cette table en écrivant la relation :

**`Note(id_note : INT, #id_eleve : INT, matiere : TEXT, valeur : FLOAT)`**

Une note est donc rattachée à un élève spécifique par le champ `id_eleve` (précédé d'un `#` pour indiquer que cette valeur **doit correspondre** à une valeur présente dans une *autre table*. Notez que l'information sur le nom de cette table n'est indiquée dans *aucune* relation. Une *bonne pratique* est donc de donner le *même nom* aux champs de différentes tables qui doivent coïncider.

⚠️ **Intégrité référentielle**

Une valeur de clef étrangère doit **toujours** exister dans la table référencée. On ne peut pas avoir une note qui pointe vers un élève inexistant, par exemple.

# 4. Anomalies dans un schéma

Un mauvais schéma peut provoquer trois types d'anomalies.

- **Redondance :** la même information est stockée à plusieurs endroits → risque d'incohérence.
- **Anomalie d'insertion :** on ne peut pas insérer une donnée sans en connaître une autre (ex. : impossible d'ajouter une matière sans y associer immédiatement un élève).

- **Anomalie de suppression :** supprimer un p-uplet détruit des informations utiles par effet de bord.
- **Anomalie de mise à jour :** modifier une information oblige à la changer à plusieurs endroits, avec risque d'oubli.

*La solution est de **décomposer** les tables pour que chaque information n'apparaisse qu'une seule fois.*

# 5. SGBD — système de gestion de bases de données

Un **SGBD** (ex. : SQLite, PostgreSQL, MySQL) est le logiciel qui gère *physiquement* la base (par *physique*, on parle de la façon dont les informations sont stockées sur un disque dur). En plus du stockage des données, il gère aussi leur interrogation, leur modification et la sécurité des accès. Il rend quatre services essentiels.

- **Persistance :** les données survivent à l'arrêt du programme.
- **Accès concurrents :** plusieurs utilisateurs peuvent lire et écrire simultanément sans corrompre les données.
- **Efficacité :** le SGBD optimise l'exécution des requêtes, même sur de grandes quantités de données.
- **Sécurisation :** gestion des droits d'accès (qui peut lire, écrire, modifier).

> ⚠️ **Attention !**
>
> En NSI, on comprend le **rôle** de ces services — pas leur fonctionnement interne.

# 6. SQL — interrogation et mise à jour

## 6.1. Structure générique d'une requête d'interrogation

```sql
SELECT attributs
FROM table
JOIN autre_table ON condition_de_jointure
WHERE condition;
```

- `SELECT` : indique les noms des colonnes dans lesquelles l'information sera puisée.
- `FROM` : indique le nom de la table dans laquelle se trouvent les colonnes précédentes.
- `WHERE` : indique une condition que les données devront être respectées (on peut combiner plusieurs

conditions avec `AND`, `OR` ou `NOT`).

- `JOIN` : quand des informations doivent être « piochées » dans plusieurs tables, il faut pouvoir les relier logiquement. La *condition de jointure* permet de préciser comment les données de deux tables seront associées (la base doit avoir été conçue pour permettre cette association entre tables) ;
- `;` final **indispensable**.

## 6.2. Jointure interne

```
SELECT Eleve.nom, Note.matiere, Note.valeur
FROM Eleve
JOIN Note ON Eleve.id = Note.id_eleve;
```

- Les noms des colonnes sont *préfixés* par celui des tables : c'est utile si deux tables *différentes* ont des colonnes qui portent le *même nom*.
- Seuls les p-uplets vérifiant la condition dans les **deux** tables sont retournés.

💭 **Remarque : explications complémentaires sur l'exemple précédent ▼**

Ici, on a une table qui enregistre *tous les élèves*, et une table qui enregistre *toutes les notes*. Il faut donc un moyen d'associer une note à un élève :

- Chaque élève est enregistré dans une table nommée `Eleve`, table qui a une clef primaire nommée `id`. C'est probablement un identifiant numérique unique, qui permet à un élève portant le *même nom* de figurer dans la table simultanément.
- Chaque note est associée à l'élève qui l'a obtenue par le biais de cet identifiant unique. Dans la table `Note`, le champ (la colonne) porte le nom `id_eleve`.
- Concrètement, cela donne quelque chose comme :

```
Eleve
id | nom     | prenom | classe
---|---------|--------|-------
1  | Martin  | Alice  | TG3
2  | Dupont  | Léo    | TG3
```

```
Note
id | id_eleve | matiere | valeur
---|----------|---------|-------
1  | 1        | NSI     | 16
2  | 1        | Maths   | 14
3  | 2        | NSI     | 12
```

## 6.3. Modificateurs utiles

```sql
SELECT DISTINCT classe FROM Eleve;          -- liste des classes, sans doublon
SELECT nom FROM Eleve ORDER BY nom ASC;     -- tri croissant (ordre lexicographique)
SELECT MAX(valeur) FROM Note;               -- la meilleure note
SELECT AVG(valeur) FROM Note;               -- moyenne (fonction d'agrégation)
SELECT COUNT(*) FROM Note;                  -- nombre de notes enregistrées dans la
```

**Les fonctions d'*agrégation* courantes sont :** `COUNT`, `SUM`, `AVG`, `MIN`, `MAX`.

> ⚠️ **Hors programme**
>
> `GROUP BY` et `HAVING` ne sont **pas** au programme de Terminale NSI. Mais un sujet peut très bien les mentionner, en expliquant brièvement leur rôle et leur fonctionnement.

### 6.4. Requêtes de mise à jour

```sql
INSERT INTO Eleve (nom, prenom, classe) VALUES ('Dupont', 'Marie', 'TG3');
UPDATE Eleve SET classe = 'TG4' WHERE id = 12;
DELETE FROM Eleve WHERE id = 12;
```

> ⚠️ **Attention !**
>
> Un `DELETE` ou `UPDATE` sans clause `WHERE` affecte **tous** les p-uplets de la table. Un `UPDATE` *modifie* des données existantes ; il est donc moins « risqué » qu'un `DELETE`, mais une mauvaise clause `WHERE` peut affecter toutes les lignes.

## 7. Liens avec d'autres notions

- La **clef étrangère** est le mécanisme concret qui donne son sens au mot *relationnel*.
- La **redondance** est l'ennemi : un bon schéma la minimise en répartissant les données dans plusieurs tables liées.
- **SQLite** est certainement le SGBD le plus utilisé en NSI — léger, sans serveur, et pilotable depuis Python avec le module `sqlite3`.